483281

RADC-TR-66-37

MATHEMATICAL MODELS OF INFORMATION SYSTEMS

Richard F. Arnold
Harvey L. Garner
Richard M. Karp
Eugene L. Lawler

TECHNICAL REPORT NO. RADC-TR-66-37
April 1966

Rome Air Development Center
Research and Technology Division
Air Force Systems Command
Griffiss Air Force Base, New York

MATHEMATICAL MODELS OF INFORMATION SYSTEMS

Richard F. Arnold
Harvey L. Garner
Richard M. Karp
Eugene L. Lawler

FOREWORD

This technical report was prepared by Systems Engineering Laboratory, Department of Electrical Engineering, The University of Michigan, Ann Arbor, Michigan under Contract AF30(602)-3546. The work was performed under Project 5581, Task 558109. The RADC program monitor is Mr. Morris A. Knapp, EMIID.

Release of subject report to the general public is prohibited by the Strategic Trade Control Program, Mutual Defense Assistance Control List (revised 6 January 1965), published by the Department of State.

This technical report has been reviewed and is approved.

Approved: [illegible]
FRANK J. TOMAINI
Chief, Info Processing Branch

Approved: [illegible]
ROBERT J. QUINN, JR.
Colonel, USAF
Chief, Intel and Info Processing Div

ABSTRACT

This report is the first interim report of a three year study and investigation by the University of Michigan. The primary objective of this effort is the study and development of mathematical models of information processing systems. The general area of research includes machine design, automata theory, and the application of mathematical models to problems in machine design. The areas of research in this report are divided into these four areas (1) Automata Theory and Applications, (2) Theory of Algorithms, (3) System Analysis, and (4) Combinatorics and Switching Theory.

Table of Contents

Preface

This report is a summary of the results obtained from the first year of research under contract AF 30(602)-3546. In this report many mathematical details and proofs have been omitted in order to obtain a short, readable document. Such detail can be found in the recent publications of the laboratory personnel. These publications are listed at the end of this report and should be considered as part of the research documentation.

The primary objective of this research is the development of models of information processing systems or models of the functional parts of information processing systems. The choice of research areas is influenced by existing problems in information processing systems and by the suitability of these problems to abstraction and modeling. The general area of research includes machine design, automata theory, and the application of mathematical models to problems in machine design.

The report is divided into four parts: (1) Automata Theory and Applications, (2) Theory of Algorithms, (3) Systems Analysis, and (4) Combinatorics and Switching Theory. In some cases the categorization of a given topic in a particular area is arbitrary. In general, Automata Theory and Applications is concerned with the behavioral aspects of finite state machines in the abstract sense and, in particular with relations between structure and behavior in the case of circuit malfunctions. The relationship of abstract machines to programming language is summarized and comments are made with respect to the need for a better understanding of semantics. The Theory of Algorithms is concerned with the question of the existence and the derivation of optimum algorithms. The abstract approach is taken. In Systems Analysis models are presented for the organization of a random access store and for concurrent computation. The results of a simulation study of the storage allocation problem are presented. Preliminary thoughts on the classification of machines and problems is included. Models of the control process are being studied but this research is in the preliminary stage and is not included in this report. Combinatorics and Switching Theory is concerned primarily with covering problems and recent results in the area of threshold networks.

A comprehensive literature survey was undertaken early in the course of this research. A detailed review of this literature is not included in this report but mention is made to pertinent references and other research efforts of interest to this program.

Professor Harvey L. Garner has served as the project director. The senior research staff has consisted of Professor Richard F. Arnold, Professor Eugene L. Lawler, and Professor Richard M. Karp. The research staff has consisted of P. Dauber, J. DiGiuseppe, K. Garrison, R. Gonzalez, J. Meyer, V. Powers, G. Putzolu, R. Reiter, D. Wood, and R. Zauel.

1. Automata Theory and Applications

1.0 Introduction

In the decade or so that automata theory has been a recognized field of study, great strides have been made in the discovery of theoretical results and in the explication of the interfaces with bordering disciplines. Among the notable achievements have been (1) a thorough analytic treatment of finite-state systems and (2) the development of an elegant and coherent theory of formal languages and of the automata that can be used to generate, manipulate, and recognize them. These two bodies of theory seem to have particularly great potential for application in the analysis and design of computer systems.

Our main research effort in finite-state systems has been concerned with error properties (aside from the question of synthesis, discussed in section 4), and two complementary points of view have been adopted. Under one point of view a malfunction is considered to be temporary, leaving the state transition structure of the machine unaltered. A natural question to ask is: "What is the probability that the system will be automatically restored to its proper state?" Under the other point of view, a malfunction is considered to be permanent, changing the state transition structure in some significant way. Natural questions to ask are: "How is the new system related to the original? Can the inputs be modified or coded sc that the desired behavior is realized in spite of the malfunction? Can the system be designed in such a way that malfunctions in the system will produce only tolerable changes?"

A study of the temporary type of malfunctions conducted by P. Dauber is reported in Section 2. The initial formulation of a study of permanent malfunctions being conducted by J. Meyer are presented in Section 1.

The theory of formal languages has contributed to a better understanding of the necessary properties of programming languages and to more systematic methods of specification and of processing. Various unsolved problems, including an inadequate treatment of semantics, appear to limit further applications in the design and programming of computers. These problems are reviewed, and an appraisal of the field is made in Section 3.

1.1 Permanent Malfunctions in Sequential Machines

The particular point of view adopted in treating the subject of malfunctions, failures or errors depends strongly on the class of errors being considered, where different error classes reflect different interpretations as to how and where the errors are caused. If the errors are regarded as arising externally, that is, at the input to the machine, then the internal structure is not affected by error. Thus, the effects of input error on the behavior of a particular sequential machine can be classified and analyzed in terms of its structure (transition function), a structure which one can assume to be fixed under all possible errors. Representative of this type of error analysis are the contributions of Neumann [b31], Winograd [b36], and most recently Harrison [b18].

Another general class of errors are those which are regarded as being the result of internal malfunctions. In the case where the cause of the error is transient or temporary in nature and results in the machine assuming an erroneous state, the analysis of such errors is closely related to that of the input-error interpretation. This class of errors can again be related to a fixed, deterministic structure since, following the disturbance, the operation is assumed to be error free. Thus, this type of error can equally well be regarded as being caused externally by some input error. The questions asked differ, however, since they relate directly to state errors rather than to the nature of the inputs that might have caused them. This is the point of view adopted by Hartmanis and Stearns [b20] and Dauber [a4] in their investigations of this type of state transition error.

The nature of internally caused errors has also been studied extensively from a synthetic point of view; that is, in terms o. ·imitive elements whose interconnection realizes some sequential network. Here, the primitive elements (which have been regarded as logic primitives and neurons at one extreme and large subsystems at the other) are assumed to have a certain probabilistic behavior. The analysis is then concerned with questions that relate the probabilistic nature of the primitive elements to that of interconnected networks of these elements. This was the point of view taken in the classic contributions of von Neumann [b35] and Shannon and Moore [b28]. Since their papers appeared, much of the effort devoted to the study of reliability and redundancy in switching networks and computing systems has reflected this particular method of analysis [b23].

The reason for introducing this short summary regarding various classes of errors and methods with which they have been investigated (the summary is in no sense complete) is to provide a means of comparing the class of errors we propose to investigate here with certain of those already studied. The type of errors we wish to consider here can be regarded as being due to internal failures which are permanent in nature. Unlike the class of externally caused errors or the class of internally caused transient errors, we can no longer assume that the structure of the machine is invariant. On the other hand, we would like to avoid restricting ourselves to a particular class of switching elements as is done in the synthetic approach. This is not to say that a more general analysis would have no bearing on synthesis problems. We feel that by relating the effects of permanent failure directly to structure one can maintain a certain degree of generality and yet provide synthesis procedures that are applicable to various classes of switching networks. We will have more to say in this regard once we have outlined the basic framework and objectives of our proposed research.

We are given a finite-state sequential machine

$$M = (\Sigma, \Delta, Q, \lambda, \delta)$$

with inputs Σ, outputs Δ, states Q, transition function δ, and output function λ defined in the usual way. We now suppose that in some physical system represented by M there is a permanent malfunction which permanently alters the system but results in a configuration that still behaves as a sequential machine. We can then represent the result of the failure as a second machine

$$M' = (\Sigma, \Delta, Q', \lambda', \delta')$$

where the states Q', the transition function δ', and the output function λ' of the failed machine are related in some way to the original machine M. A more precise statement of this relationship depends, of course, on more detailed knowledge as to how the system failed.

In relating M to M' we choose to restrict our attention here to failures that occur in the memory portion of the physical system. This restriction is motivated by the fact that it is memory which distinguishes nontrivial sequential machines from purely combinational systems. The latter have been investigated rather thoroughly with regard to error susceptibility but few of the results apply when memory (with feedback) is introduced. The restriction also has the advantage that the function of memory is the same from machine to machine, that is, to store the information presented at the memory input.

In a sequential machine the transition function represents both decision and memory processes in that we interpret $\delta(q,\sigma)$ to be the "next" state given the "present" state is q and the "present" input is σ. To distinguish the functions of memory and decision we can let $\boldsymbol{\delta} = \delta\cdot\mu$ (the functional composition of δ and μ, first applying δ) where $\delta(q\cdot\sigma)$ is the memory input and represents a purely combinational process and μ is the memory function representing the storage of $\delta(q\cdot\sigma)$. In case the memory operates properly, μ is simply the identity function on the set of states Q. Accordingly, if memory inputs are stored improperly as the result of failure, μ is some function other than the identity function. To insure that the result of the failure is stable in the sense that the machine does not oscillate between states, we require in addition that μ be an idempotent function. We call such functions <u>failures of the machine M</u> and for the failed machine M' we have $Q' = \mu(Q)$, $\boldsymbol{\delta}' = \delta\cdot\mu$ and $\lambda' = \lambda$ restricted to $\mu(Q)$. This then is the basic framework within which we intend to study permanent failures.

Within such a framework a number of interesting questions present themselves in a rather natural way. In physical situations where one is unable to or does not choose to repair the system, one is interested in failures under which the terminal behavior is unaltered by the change in structure. Thus we want to determine the conditions on a failure μ under which the failed machine is behaviorally equivalent to the original. Also of practical importance are situations where internal repair is not feasible but one is able to recode input and output information in an attempt to re-establish proper operation. This would occur, for example, with airborne or spaceborne computers that receive from and/or transmit to, manned installations. In this case we want to know the conditions on μ under which M' can simulate M. We may also ask questions about preserving certain properties of the behavior that correspond in some sense to partial success of the system.

With regard to failures themselves, we would like to determine those relations on a set of failures that will connect the known properties of some given failure with those related to it. For example, if we know that some failure μ preserves behavior in one of the above-mentioned ways, what relation or relations on the set of failures will determine other failures having the same property? Regarding the failures simply as functions, how do well-known operations and relations correspond to the intended interpretation? What subsets of failures correspond to various physically motivated restrictions as to how memory elements fail? Can these subsets be generated in some way from even smaller sets? The last two questions are related directly to the synthesis problem, which we regard as an important part of the investigation. In the proposed framework, this problem takes the following form. According to the assumed nature of the memory elements and how they fail in a memory system, one can determine, for a given size memory, the functions corresponding to possible physical failures. This can be done without knowledge as to how the remainder of the system is implemented. Then given a reduced machine which satisfies the specified behavioral requirements, the synthesis problem is translated to that of a many-to-one state assignment which satisfies the desired behavioral requirement (e.g. equivalence or simulation) on machines resulting from failures in the set under consideration. As such an assignment may not always be possible; questions as to the conditions under which algorithms exist are also important.

In the process of answering these questions we should also obtain a more thorough knowledge as to general relationships between structure and behavior, and should be able to relate failures to other aspects of machine theory, especially decomposition theory. The main objective, however, is to determine, as a function of the complexity of behavior, the complexity of structure needed for a specified degree of error insensitivity. In the case of permanent failures, such questions have not been answered by the synthetic approach, whereas we feel they can be answered within the proposed framework.

1.2 Temporary Malfunctions in Finite Automata

This problem arose from an attempt to make a general study of reliability in computer-like machines. Machines of this type may, due to a bad input tape, a temporary malfunction of a diode, or for some other reason, enter an incorrect state. The machine may then under the influence of the input tape yield incorrect outputs. However, if the input tape takes both an incorrect state, entered due to the malfunction, and the correct state to the same next state, then subsequent outputs will be correct. This will be called correcting the error.

The classic results of von Neumann [b35] apply only to networks without feedback. Thus a malfunction only causes the network to be in the incorrect state for a bounded length of time. With feedback a malfunction can cause an error which may persist forever. Fortunately, not all errors are of this type. Some errors are of the type that can persist only for a bounded time. Some errors, although they can persist infinitely long, have a probability one of being corrected as the tapes get longer. Thus "almost all" of the "long" tapes correct the error.

This phenomenon has been studied by Dauber [a5 , a6]. The results of this research are summarized here. First the problem is formulated in terms of the theory of automata. In the formulation which follows we are concerned only with states and outputs are ignored. This transition system is sufficient for this study.

Definition 1

A finite automaton M is a triple

$$M = (Q, \Sigma, \delta)$$

where Q is a finite set with elements q_i (set of states);

Σ is a finite set with elements σ_i (input alphabet);

δ is a function from $Q \times \Sigma \rightarrow Q$ (next state function).

If we are thinking of the finite automaton as a model of a digital device, we can associate the states of the digital device with the state set Q, the input symbols of the device with Σ. Then the manner in which the device changes states, when it receives an input is associated with δ.

Definition 2

(a) An error in a finite automaton, M, is a pair of states (q_i,q_j).

(b) An error (q_i,q_j) is corrected by a tape t ("tape" is synonymous with "input sequence") if and only if $\delta(q_i,t) = \delta(q_j,t)$.

We can think of an error (q_i,q_j) as the situation, when due to a previous malfunction, the automaton is in state q_i and should be in state q_j or vice versa. (It is obvious from the definition "corrected" that these situations are equivalent.) If, by the above definition, an error is corrected, then from that time on the output must be correct. However if an error is not corrected we may still have more incorrect outputs.

In this work we will consider a random source, which generates sequences and drives the automaton. A random source S is a set $\{P_n\}$ of probability distributions $P_n(x)$, the probability of the n-length string x. S has property P if there is a real $k > 0$ such that $P_{n+1}(x\sigma) \geq k\, P_n(x)$, for all x and σ.

Lefinition 3

Let S be a random source with property P and output symbols Σ, and let $M = (Q, \Sigma, \delta)$ be a finite automaton driven by S. For an error

$E = (q_i, q_j)$ we define the following:

(a) $\gamma^S_\ell(q_i, q_j)$ = probability of the set of tapes of length ℓ which correct the error (q_i, q_j).

(b) $\gamma^S(q_i, q_j) = \lim\limits_{\ell \to \infty} \gamma^S_\ell(q_i, q_j)$.

Now let us consider the following classification of errors in a finite automaton M being driven by a source S as above.

Definition 4

An error $E = (q_i, q_j)$ is

(a) definite if and only if there is an ℓ such that $\gamma^S_\ell(E) = 1$.

(b) finite if and only if $\gamma^S(E) = 1$.

(c) correctable if and only if $\gamma^S(E) > 0$.

(d) non-correctable if and only if $\gamma^S(E) = 0$.

Intuitively, these classifications have the following meanings. If an error is definite then there is a fixed length, ℓ, such that all sequences of length ℓ or greater, correct the error. On the other hand, if there is not such a fixed length, but as we consider longer and longer sequences, a larger and larger percentage of sequences correct the error, and if in the limit one hundred percent of the sequences correct the error, then we have a finite error.

We will now give some fundamental properties of errors which will show the connection between the concepts of correctable and finite errors.

Theorem 1

The set of finite errors in an automaton M driven by a source with property P induces a partition on the set of states. That is, there is a partition π_F on the states of M so that $E = (q_i, q_j)$ is finite if and only if $q_i \equiv q_j (\pi_F)$.

The next theorem will show the strong connection between errors which are correctable and errors which are finite. We will use the cross product of sets which has the following meaning:

$$A \times B = \{(a_i, b_j) \mid a_i \in A \text{ and } b_j \in B\}.$$

Theorem 2

Let $C \subset M \times M$ be the relation: $(q_i, q_j) \in C$ if and only if (q_i, q_j) is a correctable error. Then an error $E = (q_i, q_j)$ is finite if and only if $(q_i, q_j) \in C$ and for all tapes t, $(\delta(q_i, t), \delta(q_j, t)) \in C$.

Since the concept of an error being correctable is not dependent upon the source, the above theorem tells us that as long as we are dealing with only the class of sources that have property P, the property of an error being finite is also independent of the source. In the discussion which follows the term "source" will refer to a source with property P unless we explicitly say otherwise. Thus we will call an error a finite error if it is finite for some source (hence all sources) with property P, and we will call π_F the finite error partition. Likewise we will drop the superscript on γ denoting the source. We will call an error a nontrivial error if it is not of the form (q_1,q_1).

Theorem 2 provides a conceptual connection between the relation C and the partition π_F. The next theorem is a stronger characterization of this connection.

We will use the canonical ordering on partitions. That is,

$$\pi_1 \geq \pi_2 \Longleftrightarrow (q_1 \equiv q_2(\pi_2) \Longrightarrow q_1 \equiv q_2(\pi_1)).$$

Theorem 3

π_F is the coarsest partition (i.e. the largest under the canonical ordering) with the substitution property such that

$$q_i \equiv q_j \ (\pi_F) \Longrightarrow (q_i,q_j) \in C.$$

State behavior realization as defined by Hartmanis and Stearns [b19,b22] is a very strong type of realization. One meaning is that if one digital circuit is represented by an automaton M_1 and another by an automaton M_2 and if M_1 state behavior realizes M_2 then no matter what coding of the outputs we put on the second system there is a coding of outputs of the first so that they both do the same thing.

Definition 5

Let $M_1 = (Q_1, \Sigma, \delta)$, $M_2 = (Q_2, \Sigma', \delta')$ and $\lambda : Q_1 \times \Sigma \rightarrow \Sigma'$. Then the series connection of M_1 with M_2 with connecting function λ is the automaton $M = (Q_1 \times Q_2, \Sigma, \delta'')$ where δ'' is defined as follows:

$$\delta''((q_i,q_j),\sigma) = (\delta(q_i,\sigma), \delta'(q_j, \lambda(q_i,\sigma))).$$

We will say that a finite automaton M can be state behavior realized by a series connection of finite automata M_1 and M_2 if there is a connecting function λ such that M is state behavior realized by the series connection of M_1 and M_2 with connecting function λ.

Again, in terms of digital systems, this means that if D_1, D_2, and D_3 are three digital systems represented by finite automata M_1, M_2, and M_3 and M_1 is state behavior realized by a series connection of M_2 and M_3, then instead of building D_1, we can build D_2 and D_3 and connect them as follows:

$$\begin{tikzcd} {} \arrow[r, dash] & \boxed{D_2} \arrow[r] & \boxed{D_3} \arrow[r, dash] & {} \end{tikzcd} .$$

In this case λ corresponds to the coding of the outputs of D_2.

The following theorem is a consequence of Theorem 3 and a well-known result of Hartmanis.

Theorem 4

If M is a finite automaton with a finite partition π_F, then M can be state behavior realized by a series connection of two automata M_1 and M_2, where all errors in M_2 are finite, and M_1 has no nontrivial finite errors.

The following example demonstrates these theorems.

Example 1

Let $M = (\{a,b,c,d,e\}, \{0,1\}, \delta)$ where δ is the mapping shown below.

δ	0	1
a	b	d
b	a	d
c	a	b
d	b	d
e	a	d

It is easy to show that

$$C = \{(a,d),(d,a),(b,c),(c,b),(e,a),(a,e),(e,d),(d,e),(b,e),(e,b),$$
$$(c,e),(e,c),(a,a),(b,b),(c,c),(d,d),(e,e)\}.$$

There are four equivalence relations with the substitution property contained in C.

$$\pi_1 = \{\bar{a},\bar{b},\bar{c},\bar{d},\bar{e}\}$$
$$\pi_2 = \{\overline{a,d}, \bar{b}, \bar{c}, \bar{e}\}$$
$$\pi_3 = \{\bar{a}, \overline{b,c}, \bar{d}, \bar{e}\}$$
$$\pi_4 = \{\overline{a,d}, \overline{b,c}, \bar{e}\}.$$

The coarsest one is π_4. Thus the only nontrivial finite errors are

$$\{(a,d),(d,a),(b,c),(c,b)\}.$$

The following theorem is another corollary of Theorem 3. It was first proved in another context by C'lbert and Moore [b14].

Theorem 5

All errors in an automaton M are finite if and only if M has a reset tape. (A tape t is a reset tape if $\delta(q_i,t)$ is independent of q_i.).

Let us now look at another example to show the use of this theorem.

Example 2

Let M = ({a,b,c,d}, {0,1}, δ) where δ is shown below.

δ	0	1
a	b	c
b	c	d
c	d	b
d	d	b

It is easy to see that all the errors are correctable. Hence by Theorem 3, $\pi_F = \{\overline{a,b,c,d}\}$ and all errors are finite. Upon examination it can be seen that the tape 000 is a reset tape since $\delta(q_1,000) = d$ regardless of q_1. Thus we could have found that all errors are finite by applying Theorem 5.

In the course of studying errors in finite automata, it became obvious that many of the error properties (as well as other properties) of finite automata were more easily discussed in terms of the semigroups of the automata [b25, b26]. We summarize some of these results here, without repeating the exposition of abstract semigroups contained in the complete report.

Definition 6

Let M = (Q, Σ, δ) be a finite automaton. S_M, the semigroup of the finite automaton M, is the semigroup whose elements are transformations, mapping the set of states Q into itself, induced by the next state function δ. The multiplication operation $s_1 \cdot s_2$ is the composition of s_2 and s_1.

It is clear that the multiplication operation in the above definition is associative and that the set of its elements is closed under multiplication. Hence S_M is indeed a semigroup, as desired.

Theorem 6

Let E be an error in a finite automaton M. Then E is

1. correctable if and only if E is corrected by some minimum idempotent of S_M.
2. finite if and only if E is corrected by every minimum idempotent of S_M.

We will now state an immediate corollary to this theorem. We will use π_s to indicate the partition induced by the mapping associated with the semigroup element s. That is, two elements are in the same class of the partition π_s if and only if the mapping s takes them into the same element.

Theorem 7

An error $E = (q_i, q_j)$ in a finite automaton M is

1. correctable if and only if $q_i \equiv q_j(\pi_{s_i})$ for s_i, some minimum idempotent of S_M.
2. finite if and only if $q_i \equiv q_j \; (\bigcap_{\{s_i\}} \pi_{s_i})$ where $\{s_i\}$ is the set of minimum idempotents of S_M.

It follows that the partitions associated with the minimum idempotents of S_M completely characterize the error properties of the automaton M. If the partitions associated with the minimum idempotents are known, then we know which errors are correctable, which are noncorrectable, and which are finite.

Definition 7 (Perles, Rabin, and Šhamir, [b32])

A finite automaton $M = (Q,\Sigma,\delta)$ has a k-definite move function if and only if for all sequences $\sigma_1 \ldots \sigma_k$ of k letters from Σ; $\delta(q_i, \sigma_1 \ldots \sigma_k) = \delta(q_j, \sigma_1 \ldots \sigma_k)$ for all q_i, q_j in M. Note that this implies that $\delta(q_i, \sigma_1 \ldots \sigma_k) = \delta(q_j, \sigma_{k+1}, \ldots, \sigma_\ell \sigma_1 \ldots \sigma_k)$. We will informally call a finite automaton k-definite if its move function is k-definite.

The tie up of definite errors and definite automata is clearer after the following two theorems which are due to Hartmanis and Stearns [b20].

Theorem 8

A finite automaton $M = (Q, \Sigma, \delta)$ is definite if and only if all its errors are definite.

Theorem 9

If M is a finite automaton, then M can be decomposed into a series connection of two finite automata M_1 and M_2 as shown below with all errors in M_2 being definite and no nontrivial error in M_1 being definite. Hence M_2 is a definite automaton.

$$\begin{tikzcd} \boxed{M_1} \arrow[r] & \boxed{M_2} \end{tikzcd}$$

This theorem follows from the fact that there is a partition with substitution property π_D on the states of M with the property that an error E is definite if and only if $\pi_E \leq \pi_D$. The next two definitions and theorem give a characterization of the semigroups associated with definite automata.

Definition 8

Let $S = (S,\cdot)$ be a semigroup. Then, an element z of S is a right zero if and only if for all $s \in S$, $s\cdot z = z$.

Definition 9

We will say that $M = (Q,\Sigma,\delta)$ is a union of the finite automata $M_i = (Q_i, \Sigma, \delta_i)$ $i=1,\ldots,k$ if the following conditions hold:

(1) $i \neq j \Longrightarrow Q_i \cap Q_j$ is empty.

(2) $\bigcup_i Q_i = Q$.

(3) $\delta_i = \delta$ restricted to Q_i.

Theorem 10

Let $M = (Q, \Sigma, \delta)$ be a finite automaton. Then the following two conditions are equivalent:

(1) M is a union of definite automata.

(2) S_M contains a universal minimum right ideal U such that all elements of U are idempotent and all the idempotents of S_M are in U.

Since a finite automaton is a union of definite automata if and only if all its errors are definite or non-correctable we know a method of checking for this condition from the semigroup. The next two theorems which are corollaries of this theorem give us alternate checks.

Theorem 11

M is a union of definite automata if and only if every idempotent of S_M is a right zero.

Theorem 12

M is a union of definite automata if and only if the set of idempotents of S_M is a minimum right ideal.

Theorems 10, 11, and 12 can be applied to give more results on definite errors. For example, a finite automaton is such that all its finite errors are definite errors if and only if there are no idempotent elements outside its kernel. Also, it can be shown that linear automata have only errors which are either non-correctable or definite. The above results tell us something about the structure of semigroups of linear automata.

The complete paper by Dauber contains a number of extensions of the theory, which will not be discussed here. However, before concluding, a few words should be said about the applicability of the model.

There are three basic points which must be examined: the use of a finite automaton as an error model for a digital system, the definition of correctability, and the use of a random source driving the finite automaton. If we are considering synchronous digital systems with fixed memory capacity, then finite automata are good models. With memory which is extendable but bounded, they are still good models. However, if we consider the memory to be arbitrarily extendable then we would need a model with an infinite number of states. As was shown in Dauber [a6], the results given here do not carry over to this case.

The definition of a tape correcting an error also is not a bad model if we are careful to keep in mind what it means. It does not mean that all the outputs will be correct. It only says that all the outputs which occur on inputs after the correcting tape, are correct.

The use of a random source driving the automaton may or may not be appropriate depending upon the application. If we are modeling a digital coder such as might be used to code information on a space satellite gathering radiation information, then it is a good model. If, on the other hand, we are trying to model a digital computing system then it is a poor model. This is due to the fact that the input is a program which is not random but is truncated. However, the assumption of a random source at least gives us a way to start dealing with the problem. It may be possible in later work to modify these results for the case when we are driving the automaton with a structured source.

1.3 Formal Languages

Up to the present time the only aspect of programming languages which has been treated formally is syntax. The concept of semantics or 'meaning' which appears to be of fundamental importance both to the computer language designer and user has, up to this point, not been investigated. This omission, it appears, is due mainly to the fact that there does not exist, in general, a formal definition of the concept. This shortcoming must be overcome before the theory of language can hope to answer a number of important questions arising in computer languages.

The general theory of formal languages classifies languages into several types defined by the grammars which specify them. Chomsky [b6] gives the types numeric designations, with type 0 being the least structured and type 3 the most structured. Two types have been shown to be equivalent to known classical structures; type 0 to Turing machines, and type 3 to finite state machines. Types 1 and 2 are known as contextsensitive and context-free languages, respectively. The most useful representation in the case of programming languages appears to be context-free languages. The theory of context-free languages, their characterization and properties have been thoroughly studied [b6]. In recent years there has been an effort made to study programming languages within the framework of context-free languages.

Any application of the theory of formal languages to programming languages must differentiate between the reference language and its realization on a computer, or its hardware representation. In order to define these two we will quote from the ALGOL 60 report [b30]:

Reference Language

(1) It is the defining language.
(2) It is the basic reference and guide for compiler builders.
(3) It is the guide for all hardware representations.

Hardware Representation

(1) Each one of these is a condensation of the reference language enforced by the limited number of characters on standard input equipment.
(2) Each one of these uses the character set of a particular computer and is the language accepted by a translator for that computer.

The reference languages of the common problem oriented languages such as ALGOL, MAD, and FORTRAN, are context-free [b16]. This property only implies that there exists a set of rules or specifications defining the syntax of the individual strings in a problem oriented reference language which are context-free. This set of rules along with the set of symbols is a grammar for the reference language. Given this grammar we know that the process of determining whether a given string is an element of the language or not is well defined. The theories of automata and context-free languages provide us with a specification of a machine to implement this process. However any attempt to generalize this notion of membership to meaningful sequences of strings within the language is doomed to failure at present because semantics has not been brought in. Thus the theory gives us a way to check if the individual statements of the language are well formed but at present does not allow us to determine if a program is meaningful; that is, if it will execute with a given set of data.

A second area in which the theory of context-free languages has made a contribution to programming languages is ambiguity. The concept of ambiguity in the formal theory is associated with grammars. Intuitively, a grammar is syntactically ambiguous if it generates the same string in at least two distinct ways. This concept can then be generalized to languages. A language is inherently ambiguous if every grammar generating it is ambiguous [b15]. It is obvious that inherently ambiguous programming languages are to be avoided. However it is undecidable whether an arbitrary language is inherently ambiguous. This should stimulate attempts to define a large class of languages, rich enough for programming languages, in which there exist a decision procedure for inherent ambiguity.

The formal definition of ambiguity, unfortunately, does not encompass all the characteristics of what is, in general, meant by the term ambiguity in programming languages. The missing element again is semantics. What is needed is a way to determine whether or not a statement 'means' two different things, for if a statement means two different things the translating program will have to pick one and hence in some situations gives rise to a programming error. Within the existing formal framework a detailed examination of to what extent ambiguity causes programming errors is needed. This would allow a characterization of types of ambiguity within a programming language.

We next discuss the application of the theory of formal languages to the translation problem. In formal languages the translation problem is generally stated in this form: Given a language L_1 of type X and a language L_2 of type Y, is there a mapping h of type Z such that h maps L_1 into (onto, into infinite subset of) L_2 [b17] ?

Unfortunately, in practice a systems programmer is not interested in the existence of an arbitrary mapping. He is interested in the existence of a mapping which preserves the meaning. In other words, if someone writes an ALGOL program he would like it to be translated into a machine language program which does what he wants it to do. Thus once again the question of semantics arises.

Programming language designers are primarily interested in the answer to the following question: For a given language L_2 (viz. a machine language) and a given language L_1 (viz. the ALGOL reference language) can we find a language L_3 which, in some sense, is like L_2 such that there is a "good" algorithm for translating L_3 into L_1 which preserves meaning.

Work toward answering this question has just begun. So far no work has been done on translation which preserves meaning or on approximating one language with another. However there are now many people working on the question of good algorithms. The work by Hartmanis and Stearns is a good example [b21]. In their paper, they give a classification of functions according to their computational complexity. The computational complexity is a measure of the speed with which the function can be calculated by a multi-tape Turing machine. Since there is a strong correspondence between algorithms and multi-tape Turing machines, we then have a measure of for algorithms.

Fortunately, the study of formal languages has aided the systems programmer. For one thing, by studying the context-free languages and then translation, the programmer gets a much better feeling of what the translation process is like and what it should do. For instance, by the study of push-down automata, programmers have developed a whole new technique known as the push-down list. This is of great help in many translation schemes as well as for other programming purposes. The first mention of a scheme of this type appeared in 1954 in a paper by Burks, Warren, and Wright [b4]. For a more current theoretical discussion, see the paper by Schutzenberger [b34], and for a discussion of its use in various applications, see Evey [b12].

Schutzenberger formally defines a special type of automaton which is not as powerful as a Turing machine but has more power than a finite automaton. He calls this automaton a push-down automaton although his use of this term is not exactly the same as that of other authors. He then shows that the set of words recognized by these automata are unambiguous context-free languages and that there is a weak converse to this property. Evey's methods, on the other hand, are much less formal. He defines a class of machines called push-down machines. Then he shows that for every context-free language there corresponds a push-down machine that recognizes it, and conversely, the set of strings recognized by any push-down machine is a context-free language.

This brings us to a consideration of the hardware representation. In general, for a given reference language, there will be many compiler realizations of the language. Furthermore, due to the physical considerations of finiteness, each of the compiler languages is only an approximate realization of the reference language. These approximations, in general, amount to a truncation, thus giving a finite language. However the full extent of the restrictions that a compiler places on a reference language is not known in general. It is an open question to what degree the theory of context-free languages will be useful for compiler languages. Further investigation along these lines is certainly called for.

In summary, then, the situation is as follows. The theory of formal languages today is only of limited use to someone writing compiler languages. As the theory is extended to more class of languages, particularly those intermediate between context-free languages and finite automata languages, it will become more and more useful. Also, if the syntactic theory were extended to include some of the rudimentary concepts from semantic theory, there would be even more areas of potential application.

2. Theory of Algorithms

2.0 Introduction

The theory of algorithms relates to computer programs in the same way automata theory relates to the structure and the behavior of computers. (We interpret the word algorithms in a broad sense, without limiting our attention to formal systems of the Markov type.) The theory of algorithms encompasses such questions as: How should algorithms be formally characterized? What representations are appropriate? How do the computational complexity and storage requirements of a given algorithm vary with the problems in its domain? Does a given algorithm terminate for all problems in its domain? Is the result of applying an algorithm unique for all problems in its domain? When does an optimal algorithm exist (for some reasonable definition of optimality)? How can an optimal algorithm be determined?

Our program in this area encompasses three projects:

(1) The study of "monotone congruence" algorithms by R. F. Arnold and D. L. Richards. A technical report has been issued and journal publication is expected.

(2) The formalization of discrete dynamic programming algorithms, and the investigation of its connections with automata theory, by R. M. Karp.

(3) The study of "branch-and-bound" algorithms, being carried on by E. L. Lawler and D. E. Wood, under principal sponsorship of National Science Foundation Grant GP-2778.

2.1 Monotone Congruence Algorithms

Given an associative system (semi-group) in which each element is assigned a cost and in which an equivalence relation obtains between elements, it is often of interest to ask the question: What is the least costly word equivalent to a given word? Three examples of such problems are the travelling salesman problem studied by Dantzig, et al. [b8], the optimum control problem (cf. Pontryagi [b33]), and the problem of finding the least word which performs a given mapping upon the states of a finite automaton. Algorithms do not exist for all such problems, since the solution may, in general, require the solution of the word problem for semi-groups, which is known to be unsolvable (Davis [b9]).

The problems mentioned above involve natural two-sided congruence relations. Two input words x and y to a finite automaton may be regarded as equivalent if they perform the same mapping from the set of states of the automaton to itself; i.e., if $M(s,x) = M(s,y)$ for all states s where M is the transition function of the automaton. A simplified model of computer programming can be obtained by reinterpreting the input words as programs, and the initial state as data upon which the programs act; here the programs are not self-modifying and contain no instructions which transfer control. Programs P and Q are then equivalent if they reach the same result for each set of data; in that case the programs RPS and RQS must also be equivalent, R being run immediately before P (or Q) and S being run immediately after. Thus the equivalence relation satisfies the definition of a two-sided congruence.

The programming and automaton problems can be solved by enumeration if necessary; however, the solution algorithms are often impractical and may give no intuitive insight into the structure of the optimizing process itself. It is hoped that greater insight and more effective algorithms can be found by studying the general class of associative systems with two-sided congruence relations, a class which contains problems of all degrees of difficulty.

The theory of monotone congruence algorithms applies if a two-sided congruence relation exists and there exists a total order relation , on the costs of the words, which has certain natural properties. The properties of this class of algorithms has been investigated [a2] and a general result concerning order relations on finite alphabets established.

A Markov normal algorithm on a set of words Σ^* is a finite ordered list of substitutions of Σ^*. For an order relation and an equivalence relation, a monotone congruence algorithm is a Markov normal algorithm A for which:

(1) A(w) is minimal whenever it exists.

(2) for each substitution $x \rightarrow y$ of A, $x > y$ and $x \sim y$.

In terms of computer programming, the words may be programs or systems of subroutines, the letters of the words, instructions or subroutines. The order relation relates the cost of a program, i.e. in terms of execution time, space required, etc. The equivalence relation equates programs which perform the same action on the internal states of the computer. A monotone congruence algorithm is then a succession of replacements of a subprogram with a less costly one which computes the same function.

The first principal result established is that for any monotone congruence algorithm A and any word w, A(w) exists; i.e., that every monotone congruence algorithm always terminates. The proof depends on a quite general theorem about total order relations on Σ^*, i.e.: Every total order which is a refinement of the inclusion partial ordering is a well-ordering.

Auxiliary letters are parts of words which appear only in the intermediate stages of the application of an algorithm. Concluding substitutions of an algorithm are those which terminate the process immediately, without exhausting all possible other steps. It is proved in [a2] that every monotone congruence algorithm is equivalent to one which has neither auxiliary letters nor concluding substitutions.

The second principal result is that every class of mutually equivalent monotone congruence algorithms contains a unique minimal algorithm, called the core algorithm, which can be obtained by applying any member of the class to itself. The algorithm is unique up to the order of the substitutions and minimal with respect to the number of substitutions. A uniformly optimal algorithm is one which requires the fewest number of steps, or applications of the algorithm, to terminate. The study of optimality reduces to the study of the algorithms which can be produced by reordering the substitutions of the core algorithm. It has been shown, however, that a uniformly optimal ordering of the substitutions does not always exist, and that more than one uniformly optimal ordering may exist.

2.2 Discrete Dynamic Programming

Over the past several years dynamic programming has emerged as an important computational tool for the solution of multistage optimization problems. It is, therefore, surprising that the technique has never been precisely defined, and that the Principle of Optimality, which underlies dynamic programming, has not been given a precise and general statement.

R. M. Karp, together with M. Held of International Business Machines, Incorporated, has undertaken to construct a mathematical theory encompassing the domain of discrete, deterministic dynamic programming.

The two central concepts of this theory are the discrete decision process and the sequential decision process. A discrete decision process

$$D = (A,S,P,f)$$

is specified by A, a finite set of primitive decisions, S, a subset of all finite sequences of decisions, P, a space of parameters (i.e., problem data) and a real-valued cost function f with domain $S \times P$. The elements of S represent feasible policies, and the problem is to construct an algorithm to find, for any given $p \in P$, an element $s \in S$ minimizing $f(s,p)$. A sequential decision process is specified by a finite automaton $\mathfrak{a}$ with input alphabet A which recognizes the "event" S, a parameter space P, and a function h with the following interpretation: if the parameter specification is p, and state q of $\mathfrak{a}$ has been reached by an input sequence x having cost ξ, then the cumulative cost after the further application of input a is $h(\xi,q,a,p)$. The process is monotone if h is an increasing function of ξ.

Thus, a sequential decision process consists of a finite automaton together with additional cost structure. The minimization problem for a sequential decision process is that of finding an input sequence in the set S having minimum cost. A principal result of the theory is that if the sequential decision process is monotone, this minimization problem reduces to the solution of a system or recurrence equations of the type associated with dynamic programming; and, conversely, every such system of recurrence equations can be associated with the minimization problem for some monotone sequential decision process. A second principal result is the characterization of the possible representations of a discrete decision process by a monotone sequential decision process having the same minimization problem. This problem is akin to that of relating behavior to structure in finite automata, and similar methods are used.

Thus, we feel that a useful and novel coupling of automata theory with optimization theory has been achieved. It is our conviction, based on the examination of many examples, that any dynamic programming formulation of the optimization problem for a discrete, deterministic process can faithfully be interpreted as the representation of that process by a monotone sequential decision process.

2.3 Branch-and-Bound Algorithms

Among the most general and most useful approaches to the solution of constrained optimization problems is that of "branching-and-bounding" [a14] (or, according to Bertier and Roy [b2], "separation et evaluation progressives"). Most commonly, this is a technique under which the space of feasible solution is repeatedly partitioned into smaller and smaller subsets, and a lower bound (in the case of minimization) is calculated on the cost of the solutions within each subset. When the bound for any subset exceeds the cost of a known feasible solution, it follows that no solution within the subset can be optimal. The partitioning continues until a feasible solution is found such that its cost is no greater than the bound for any subset.

There has been some recent interest in branch-and-bound methods for integer linear programming and for the traveling salesman problem. These algorithms should perhaps be characterized as "meta-algorithms,", since they represent methods by which known algorithms can be embedded in a higher level procedure which can be applied to solve problems outside the domain of the original algorithm. For example, suppose we are confronted with a "difficult" constrained optimization problem of the form

$$\text{minimize } c^{(0)}(x)$$

subject to

$$g_1^{(0)}(x) \geq 0$$

$$g_2^{(0)}(x) \geq 0 \qquad (0)$$

$$\vdots$$

and $$x \in X^{(0)},$$

where $X^{(0)}$ denotes the permissible domain of optimization, e.g. the positive orthant of Euclidean n-space. (We let x denote a vector $(x_1, x_2, \ldots, x_n)$.) The problem is "difficult" to solve directly, possibly because the objective function or constraints are nonconvex or because some or all of the variables are restricted to discrete values.

Under the branch-and-bound approach the original problem is replaced by a number of "bounding" problems. We refer to the original problem as problem (0), and denote the bounding problems which replace it as problems (1), (2), (3), ... etc. The cost function, constraints, and domain of optimization of problem (j) are labelled with superscript (j). Thus, problem (j) is of the form

$$\text{minimize } c^{(j)}(x)$$

subject to

$$g_1^{(j)}(x) \geq 0$$

$$g_2^{(j)}(x) \geq 0 \qquad (j)$$

$$\vdots$$

and $$x \epsilon X^{(j)}.$$

A definition of the branch-and-bound process can be given inductively. That is, given a set of bounding problems at some intermediate point in the calculations, it is specified how a new set of bounding problems is obtained by "branching".

We let the current set of bounding problems be denoted (1), (2), (3), ..., (p). In order to be a valid set of bounding problems, it is sufficient that the following <u>bounding</u> requirement be satisfied:

(B) If x is a feasible solution to problem (0), then there exists a bounding problem (j) for which x is feasible and $c^{(j)}(x) \leq c^{(0)}(x)$.

The bounding problems are, by assumption, "easy" problems, and can be solved by appropriate direct methods. Let the optimal solutions thereby obtained be denoted $x^{(1)}$, $x^{(2)}$, ..., $x^{(p)}$, respectively. The solution $x^{(k)}$ is optimal for problem (0) if

(S1) $c^{(k)}(x^{(k)}) \leq c^{(j)}(x^{(j)})$, for $j=1,2,\ldots,p$

(S2) $x^{(k)}$ is feasible for problem (0)

(S3) $c^{(0)}(x^{(k)}) = c^{(k)}(x^{(k)})$.

A solution $x^{(k)}$ which satisfies (S1), (S2), and (S3) solves the original problems. "Branching", i.e. replacing one of the existing bounding problems by two or more new bounding problems, must continue until a solution which satisfies (S1), (S2), and (S3) is found.

The work under this project has encompassed:

(1) A survey of branch-and-bound algorithms as applied to various

types of optimization problems.

(2) The development of new branch-and-bound algorithms for problems of interest.

(3) An investigation of optimal strategies [illegible]r branching. I.e., strategies which minimize the len[illegible] [illegible]f computation or the amount of computer storage.

(4) The development of a stochastic model for branching algorithms. It is hoped that this model can be used to predict when branch-and-bound methods can be successfully applied and when they cannot: i.e., which distributions of costs in the domain of feasible solutions are favorable, and which are not.

3. Systems Analysis

3.0 Introduction

The development of models for the analysis and study of the functional parts of information processing systems and the classification of information processing systems and problems are the objectives of this part of the research program. The model of a functional part of an information processing system should permit comparisons to be made between known solutions and should suggest new solutions. This research is abstract but practical problems in information processing systems design have a direct influence on the formulation of the models and the questions asked.

Major emphasis is given to algebraic models though statistical models and simulation are not excluded from the study. Automata theory models are used when applicable. Models have been developed to study problems associated with storage, multiprocessor assignment, and control. R. F. Arnold has developed a model for the study of the addressing format for random access storage. Finite state machines are used in the formulation but statistical techniques are used in the evaluation. R. Jump has simulated the dynamic allocation of storage. Karp and Reiter [a12,a13] have extended and generalized a graph model of concurrent computation due to Karp and Miller [b24].

Control models emphasizing topology and quantity of control information have been formulated by H. L. Garner and have shown the need for a more definitive relationship between the rate of computation and the rate of control information, which is now being studied. Classification and complexity studies are still in the formulative stage.

3.1 Storage Format Characterization for Random Access Storage

The large random access store or file is an important part of existing computer systems and will continue to be important in the future because of the continuing requirement for large quantities of reference data. Practical problems exist with respect to the efficient utilization of such a store. In general, it is desirable to minimize the number of accesses to the store required to obtain a given record stored in the file and to maximize the number of filed words in the store. It is not feasible, in general, to set up a one-to-one correspondence between A, the set of hardware addresses of the store and the K, the set of keys which are the identifying part of each record. The difficulty stems from the fact that not all keys are active and the active keys are not usually distributed uniformly over the key set. A sorted file provides a poor solution to this problem since a file sort is required whenever new records are added. The open file concept provides a reasonable solution. In a general open file system there exists a deterministic algorithm which generates a sequence of store addresses for each key. The record is stored in the first empty storage location associated with the set of addresses generated by the algorithm. The algorithm computes f, sometimes called a hash function, which maps K into A. Thus $f\colon K \to A$. In general, f is a many-to-one mapping. Ideally, f should map the same number of keys to each element of A but adjacent keys should be mapped into separate addresses in order to break up clusters of active keys. An error correcting group code has the property that any code word is at least a distance d from any other code word. A decoding function g maps the set of all possible received code words onto the set of messages; $g\colon R \to M$. Two elements of R which map onto the same element of M are separated by at least a distance d and an equal number of elements of R are associated with each element of M. Thus g is a useful hash function and the theory of codes provides the information required to design this type of hash function. Number system conversion algorithms have also been considered for the generation of hash functions. Hash addressing has been effectively applied [bl] to obtain a store organization for list processing. Other applications appear possible but these have not been studied in detail.

A general characterization of the storage format problem for random access stores has been obtained by R. F. Arnold and some examples analyzed. In the type of system considered, there is a conventional random access store and a finite automaton which plays the role of "reader-interpretor".

A random access store is given by a pair of sets A and W. The "state" of the store is given by an arbitrary function $c: A \rightarrow W$ termed the <u>contents function.</u>

A <u>format</u> for the representation of functions from the set X into the set Y is given by a finite automaton with the state set S, alphabet W and transition function $M: S \times W \rightarrow S$. The "output" of the automaton is given by a map $\varphi: S \rightarrow A \cup Y$. Also defined is an "initial state" function $\psi: X \rightarrow S$.

A format determines a mapping from the states of the store (A,W) into the set of partially defined functions from X into Y. Let c be a contents function; then $f_c: X \rightarrow Y$ is defined as follows. We first define a partially defined function λ_c from S into Y.

If $\psi(s) \in Y$ then $\lambda_c(s) = \varphi(s)$.

If $\varphi(s) \notin Y$ then $\lambda_c(s) = \lambda_c(M(s,c(\varphi(s))))$.

If $\lambda_c(s)$ is not determined by either of the above, then it is left undefined.

Then $f_c(x) \overset{\text{def}}{=} \lambda_c(\psi(x))$.

In a sense, this is similar to the repeated use of hash functions. Here, if the output of the automaton is not in the range of the function, the transition function is applied to obtain a new state whose output may be.

One initial problem is that of determining for some given function from X to Y and a given format, a contents function which "represents" it. Practical application requires that the representative be easily computed and that given a representation of one function, a representative of a "neighboring" function be very easily computed. By a neighbor here we mean a function whose values differ for only a few arguments. Also of interest is the obtaining of useful characterizations of the class of functions represented by a given format.

This section gives examples of certain formats where very satisfactory information can be obtained. All of these have the property that the structure of the automaton is pseudo-probabilistically determined which then justifies certain probabilistic arguments as to the existence of representatives and the computational and information theoretic efficiency of the formats.

The automata in general "contain" hash functions. In fact, this entire work can be considered as a certain natural extension of the hash function technique in which the primary disadvantage of hash addressing, namely the many-to-one character of the hash function, is obviated by artificially enlarging the range of the hash functions.

The work to date appears to provide significant improvements in handling a wide variety of content-addressability problems where the read to write ratio is fairly high.

Of more abstract interest is the relation between the structure of the automaton and the difficulty of the optimum representative computation. For certain formats this computation is trivial while for others it appears to become arbitrarily difficult.

We now consider three examples which are specific realizations of the general formulation for the random access store.

The linear functional format is the most efficient with respect to storage space at the expense of requiring extensive computations to compute the appropriate representation of a function.

For this example the range of the function $Y = Z_2 = \{0,1\}$ and a store with n-bit words is used so $W = (Z_2)^n$. The binary words of storage are treated as vectors over the mod 2 field.

Let T be an auxiliary "copy" of $(Z_2)^n$. The hash functions employed map from, X, the domain of the function, into the product set AxT, where A is the set of store addresses. For the i-th hash function

$$h_i(x) = (a_i(x), t_i(x)) \qquad i=1,2,\ldots$$

The contents for c assigns a particular element of W to each store address $a \in A$; $c(a) = w \in W$. Assume c has been determined. If $c(a_j) \neq 0$ and $c(a_i) = 0 \quad i < j$, then $f_c(x) = c(a_j(x)) \cdot t_j(x)$ (dot product)

$$= w \cdot t_j(x) \in Z_2.$$

If $c(a_i) = 0$ for all i, then f(x) is undefined.

For a given function $f\colon X \rightarrow Y$, we now consider the process by which a representative c is computed, if it exists.

Consider the subsets in X which are the inverse images of all a under h_1. For each location $a \in A$, either a $w \in W$ exists or it does not such that $w \cdot t(x) = f(x)$ for all x in the inverse image of a. If no such w exists c(a) is set equal to zero, and all elements in the inverse image are assigned new addresses using h_2. It is now necessary to recheck, for all x, $f_c(x) = w \cdot t(x)$ since new x's may have been added to the image set of any $a \in A$. This will result in more a's being assigned contents zero, and create certain new "unhoused" x's. If this process terminates, then clearly, a representative c has been found, while if it does not, no such c exists. This last statement has not been proven; however, t e c derived by the above process can be shown to be uniformly optimal in the sense that the set of a's whose contents is zero in any representative c contains the set of a's set to zero by the above process. Is the probability that a vector exists in $(Z_2)^n$ which satisfies m randomly selected linear constraints the principal information necessary to quantitatively analyze the feasibility of this storage format? A lower bound for this probability is given by the probability that m vectors in $(Z_2)^n$ are linearly independent, which is $p(m,n) > 1 - 1/2^{n-m}$.

Observe that p(m,n) is dependent only on (n-m). The number of constraints that can be applied to a given location can be increased without a decrease in the probability that the constraints can be satisfied if n-m is held constraint. Thus a requirement for x additional constraints require x additional bits per word in store.

If a representative c exists, then there exists an average inverse image size for each address location. Furthermore, if $\check{A}$ is large and the hash functions are well behaved, the probability distributions of address inverse image size will be Poisson with mean λ. It is then possible to write equilibrium equations which must hold under the assumption that the computation process terminates.

Let ρ be the proportion of locations set equal to zero.

$$\rho = \sum_{m=0}^{n-1} e^{-\lambda} \frac{\lambda^m}{m!} \left(\frac{1}{2^{n-m}}\right) + \sum_{m=n}^{\infty} e^{-\lambda} \frac{\lambda^m}{m!}$$

An ideal hash function assigns $\bar{\bar{X}}/\bar{\bar{A}} = \mu$, to each location a. We must also have:

$$\lambda = \mu + \rho\mu + \rho^2\mu + \ldots = \frac{M}{1-\rho}.$$

These relations form the necessary basis for study of the process.

The <u>hash format</u> is closely related to the conventional technique of storing in each location both the x and f(x) values. The technique does nothing more than replace x with its hash image, which at once removes the structure from X and in so doing saves space.

This format is defined as follows. We require an auxiliary set T, whose size must be decided upon by a consideration of other parameters of the problem. The set W is interpreted as (TxY) ∪ Q where Q is a set of one or more elements which are used to indicate that the standard format interpretation does not apply and the next location in the sequence must be examined.

The hash functions used map X into A $\times$ T.

$$h_i(x) = (a_i(x), t_i(x)).$$

Let

$$c(a_i(x)) \neq q \text{ and } t_i(x) = t^*(a_i)$$

the * is used to indicate values stored. $t^*(a_i)$ designates the tag part of the contents of a_i, and let i be the least such i for which this holds. Then

$$f_c(x) = y^*(a_i)$$

where $y^*(a_i)$ indicates the portion of the contents of a_i from the set Y. If no such i exists, then $f_c(x)$ is left undefined.

More informally, the reading algorithm consists in looking in successive locations, continuing as long as one finds either a "q" or a hash tag which does not match.

A computation of representatives for this process is fairly straight-forward. One "stores" the values one at a time, keeping track in a separate store, of the set of x's mapping into each address so far. For each $x \epsilon X$, one computes $h_1(x)$ and if nothing yet has been mapped into $a_1(x)$, the pair $(t^*(x), y(x))$ is stored there. If some $x' \epsilon X$ has already been mapped into the locations resulting in $(T(x_1), y(x_1))$ having been stored there, compare $t^*(x')$ with $t(x)$. If they are different, go on to $h_2(x)$, repeating the above process. If $t_1(x) \neq t^*(x')$ then not only must one continue onto $h_2(x)$ but also the contents of $a_1(x)$ must be set equal to q and x' must be put back into the set of as yet unassigned x's. If $a_1(x)$ already is set to q, go on to h_2.

For a given size store and given $\bar{\bar{X}}$ and $\bar{\bar{Y}}$, it is of interest to consider how the store should be partitioned so as to yield the fewest number of seeks per reference. Clearly, as T is made large, $\bar{\bar{A}}$ is reduced resulting in more references to each address. On the other hand, each reference that is made is less likely to result in the match of hash tags for different x's and hence the "loss" of a location as a value storing slot.

In a manner similar to that used in the linear format, a pair of equations may be written describing the equilibrium state of the writing process, which must be analyzed numerically to obtain quantitative information about the process and the optimum value of the parameters stored.

The multistate reader format technique is of particular interest for representing functions with highly unbalanced distributions on their ranges.

For this example let $Y = \{a,b\}$ and $W = \{0,1\}$. The hash functions used map from X to A.

Let $c(h_i(x)) = 0$ and $c(h_j(x)) = 1$ $\quad j < i$.

Then $f_c(x) = a$ iff $i \equiv 2 \bmod 3$

b iff $i \not\equiv 2 \bmod 3$.

If $c(h_i(x)) = 1$ for all i, $f_c(x)$ is not defined.

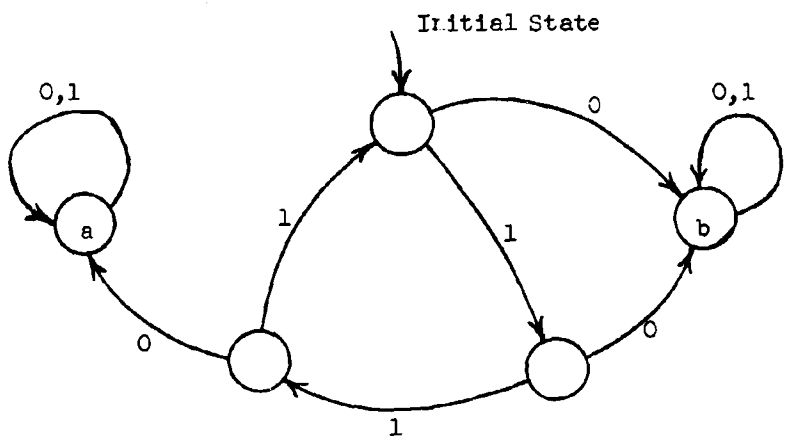
The process of computing a representative for a given function is fairly easy, and closely resembles that used with the linear format.

Start out with all locations set equal to 0. For each $x \in X$ in turn, set equal to 1, those locations necessary for that x to be correctly mapped. It will be necessary to go through the x's several times until one gets all the way through without making any changes in c. The resulting c is uniformly best since at each step we only set equal to 1 those locations which must be set to 1 for any representative. Of course, this process may not terminate in which case no representation exists.

One generalization of the above format is to replace the modulo three counter with arbitrary $n \geq 2$, and map the n congruence classes arbitrarily into some Y. Any such format will have essentially the same representative computation procedure as the above.

For large n and all but one congruence class mapped into the same element $y^* \in Y$, it is clear that if c were chosen at random, the probability that $f_c(x) = y^*$ is very large. Hence, "most" of the contents functions represent functions which are highly unbalanced on their range. While this is not in itself a proof that such a format is appropriate for representing such functions, it is a necessary condition and hints at how such a proof might go.

Still further generalization of the above format suggests replacing the 3 counter with an arbitrary finite automaton. In one such generalization the automaton is a 5-state machine in which two of the states are terminal and have output labels from Y. Observe that events are associated with each element of Y



$$E_a = (111)^* \, 110$$
$$E_b = (111)^* \, (0 \cup 10)$$

which represent the first arrivals to states having the respective labels. In general, any finite automaton having input alphabet W and some of its states labelled with elements of Y, serves to define a format.

Additional details and extensions can be found in R. F. Arnold [al].

3.2 Dynamic Allocation of Storage

A set of procedures for dynamic allocation of storage known to have been considered for implementation in an important commercial system, were chosen for evaluation. By means of simulation, it was found that the procedures were not significantly different in their performance; hence, the most easily implemented procedure within the class should be chosen.

It was assumed that the memory is divided into pages, and that the pages are assigned to, and released from, programs as needed. Each program requests and releases small blocks of contiguous words within those pages assigned to it. Blocks being used by a program are called active blocks, and those blocks that are not being used but which are within one of the pages assigned to a program are called inactive blocks.

When a page is first assigned to a program, it is considered to be a single inactive block. As the program uses it, it becomes broken into smaller blocks, both active and inactive. New active blocks are created from within old inactive blocks. If there is no inactive block within any of the pages assigned to the program which is large enough to satisfy a request for a new active block, a new page is assigned to the program. When a block becomes inactive, it combines with inactive blocks on either side to form a single inactive block. When the last (highest numbered) page to be assigned to the program becomes inactive, it is released from the program. The object of the allocation procedures is to minimize the average number of pages assigned to the program.

There are two somewhat incompatible heuristics that can be invoked. One is to fill each new request with the smallest inactive block larger than the request, without regard to the page in which this block is located.

The reasoning here is that the larger inactive blocks should be conserved, in order to reduce the probability that a future request cannot be filled with any existing inactive block. The other heuristic is to fill each request with the first (lowest numbered) available inactive block larger than the request, without regard to its size. The reasoning here is that one should use the lowest numbered blocks, in order to increase the probability that the higher-numbered pages can be released in the near future.

A compromise between these two heuristics is obtained by separating the pages into two sets. The lower-numbered pages are assigned to the first set, and the "smallest block" heuristic is applied to them. If a request cannot be filled within the first set, an attempt is made to fill it from within the second set, using the "first block" heuristic. The pages can be separated into the two sets according to a fixed or a computed proportionality factor.

The simulated programs had block lengths that were determined by weighted sums of normal distributions, and block lifes that were exponentially distributed. An efficiency factor was computed, where

$$\text{EFF} = \frac{\text{mean number of active words}}{\text{mean number of active words plus inactive words}}$$

The results of the simulations showed that efficiences as high as 0.80 can be obtained for some types of programs considered. The average efficiency for all types of programs simulated was 0.63. More important is the insignificant difference in efficiencies between the various procedures for a given program. Hence one concludes that the most easily implemented procedure is the one to be preferred.

Another question of interest is the effect of page size on the efficiency of these procedures. This was evaluated by simulating several procedures with a program which had Poisson distributed block lengths and exponentially distributed block lives. The mean block length and life were selected so that the average number of active words was relatively constant.

As the ratio of length to page size approaches zero, the average efficiency of these procedures stabilizes at approximately 0.85. The boundaries created by pages do reduce the efficiency when the page size is less than several times the average request length. However, with pages as small as two times this average request size, the efficiency is only reduced by approximately twenty percent.

These results, and extensions, are contained in a forthcoming laboratory report by J. R. Jump.

3.3 Graph Model of Concurrent Computation

It is necessary to formulate languages in which parallel computations can conveniently be described and to have methods of determining how much concurrency the intrinsic structure of such a computation allows. Karp and Reiter [a 12] have studied and generalized a model for a certain class of parallel computations. This model, originated by Karp and Miller [b24], represents a computational algorithm as a finite labelled directed graph. Each node represents an operation, and each branch represents a first-in, first-out queue of data. The sequencing of the calculation depends on four parameters (each a nonnegative integer) associated with each branch. For a typical branch p, directed, let us say, from node i to node j, the parameters are as follows: A_p, the number of data words on the branch at the beginning of the calculation; U_p, the number of words placed on the queue whenever the operation O_i associated with node i terminates; W_p, the number of words removed from the queue whenever the operation O_j initiates; and T_p ($T_p \geq W_p$), a threshold giving the minimum queue length that must be reached before O_j initiates. Each node initiates when the number of words on each of its input branches is at least equal to the corresponding threshold. With each node is associated a fixed function specifying the manner in which the inputs determine the outputs. Because of the first-in, first-out queue discipline and the absence of conditional transfers, not all calculations can be expressed as computational graphs. It appears, however, that a large class of iterative calculations can be so expressed. The computational graph model is asynchronous, in the sense that the speeds of the operations associated with the nodes are variable and unspecified. Thus, very many sequences of operation are consistent with a given graph. Nevertheless, it is shown in [b24] that the results computed in all these sequences are the same; i.e. computation graphs are determinate. Theorems are also given in [b24] which characterize the computation graphs that represent terminating computations, and provide an analysis of the growth of data queues.

A principal goal of the present effort is to extend the above model as far as possible while preserving determinacy. It is shown that determinacy is preserved in certain cases even when the operation associated with a node, as well as the parameters associated with the branches, are variable. Also, a complete analysis of termination and queue growth is carried out for the particular case in which the parameters U_p and W_p vary periodically.

In [al3], a synchronous version of the computation graph model is studied. An execution time t_i is associated with each node i, and a determination is made of the fastest periodic rate at which a computation so described can proceed. This rate may be viewed as an absolute limit on the parallelism possible in a computation.

This generalized model is better suited as a description of certain classes of computations, particularly iterative ones. As an example, the computation graph for matrix multiplication is considerably simplified under this general model.

The following problems have been considered. (1) Let G be a strongly connected computation graph in which, for all branches d_p, we have $T_p = U_p = W_p = 1$. We wish a proper execution of G which is periodic in the sense that if a node n_j first initiates at time t_j, it will initiate thereafter at times $t_j + \pi$, $t_j + 2\pi$, ..., where π, the period, is the same for all nodes of G. Clearly, if the computation is to be controlled by a clock signal, such a proper execution is desirable. Such an assignment of times t_j to nodes n_j is given. Moreover, with the proper choice of π, the computation proceeds at a maximal asymptotic rate. (2) Suppose the frequency of the clock signal controlling the initiations of the nodes of G is a priori specified. Then we wish the t_j's, and π of (1) to be integers. It turns out that if π is not an integer, the schedule of (1) will not yield integer initiation times. To aid in the investigation of a periodic schedule in this case, the so-called free running execution has been studied. This is the maximal rate of execution of a synchronous computation graph (one whose initiation times must all be integers). The major results are as follows:

(a) Let $\vec{b}(t)$ be the vector of branch weights of G at time t.

Then there exist t', λ both integers such that for all $t \geq t'$, $\vec{b}(t+\lambda) = \vec{b}(t)$.

(b) Let α_j be the number of initiations of node n_j in the period $[t, t+\lambda]$, $t \geq t'$. Then α_j is independent of j, $\alpha_j = \alpha$.

(c)
$$\frac{\lambda}{\alpha} = \pi = \max_{\text{loops of } G} \left\{\frac{\Sigma\ \tau}{\Sigma\ A}\right\}$$

where $\Sigma\ \tau$ is the sum around a loop of G of the execution times of the loop nodes and $\Sigma\ A$ is the sum of the initial numbers of data items on the loop branches.

From these results we infer that each node n_j of a synchronous computation graph under a free running execution initiates ultimately at times

$$\begin{array}{l} t_j^1,\ t_j^1 + \lambda,\ t_j^1 + 2\lambda,\ \ldots \\ t_j^2,\ t_j^2 + \lambda,\ t_j^2 + 2\lambda,\ \ldots \\ \quad \cdot\ \cdot\ \cdot\ \cdot\ \cdot \\ t_j^\alpha,\ t_j^\alpha + \lambda,\ t_j^\alpha + 2\lambda,\ \ldots \end{array}$$

where

$$t' \leq t_j^1 < t_j^2 < \ldots < t_j^\alpha < t_j^1 + \lambda.$$

(3) For a synchronous computation graph we sought an assignment of integer and periodic initiation times to the nodes of G. In the case that

$$\pi = \max_{\text{loops of } G} \left\{\frac{\Sigma\ \tau}{\Sigma\ A}\right\}$$

is not an integer, this is impossible. However, the following assignment is possible. Let $\pi = \frac{\lambda}{\alpha}$. We can find integers t_i, Δ_i^1, Δ_i^2, ..., $\Delta_i^{\alpha-1}$ such that node n_i can initiate at times

$$t_i,\ t_i + \lambda,\ t_i + 2\lambda,\ \ldots$$
$$t_i + \Delta_i^1,\ t_i + \Delta_i^1 + \lambda,\ t_i + \Delta_i^1 + 2\lambda,\ \ldots,$$
$$t_i + \Delta_i^{\alpha-1},\ t_i + \Delta_i^{\alpha-1} + \lambda,\ t_i + \Delta_i^{\alpha-1} + 2\lambda,\ \ldots,$$

This assignment is such that G computes at the maximal asymptotic rate.

It would be of interest to carry out further studies in which the computation graph model is generalized to accomodate a wider class of computations. Consideration should also be given to the problems connected with programming, for a given multiprocessor, a comp: .ation specified by a computation graph. In particular, the following questions arise:

(1) If we view each node of a computation graph as a computer capable of performing any of the node operations of G, what is the minimum number of such computers required under the varicus schedules of the problems (1), (2), and (3) just listed? More generally, if we partition the nodes of G as to their functions, what are the minimum numbers of nodes of each type required?

(2) The allocation of storage registers for data queues and their utilization so as to minimize the number required.

(3) Scheduling problems of the following type: Suppose a computation graph G requires m_i nodes of type i, i=1,2,...,n but only $m_i' < m_i$ are available. How can be constrain G so as to utilize just m_i' nodes of type i, i=1,2,...,n in an optimal fashion; i.e. at a maximal computation rate consistent with the constraints?

(4) Programming for such a computer system.

(5) The synthesis of a computation graph given a computation to be performed.

3.4 Classification of Machines and Problems

It is desirable to have measures of problem complexity and machine capability which permit relative comparisons between different machine algorithm configurations. This is particularly difficult because it is necessary to determine the proper weighting to be given to the various performance parameters such as cost, rate of computation, size, convenience, etc., which must be considered in the choice of a given machine. The real world situation is further complicated because it is usually not possible to specify a single important problem for a specific machine since in the usual situation a given machine is assigned many types of problems. For a specific algorithm it is possible to determine the maximum degree of parallelism, the minimum number of concurrent time steps required, the maximum number of time steps required and the maximum amount of storage required. A different algorithm which computes the same problem may have different requirements. For any specific machine problem configuration there is an optimum algorithm which is dependent on the machine, the problem, and the weights assigned to the performance function. Comparison of machines in a real environment is difficult because the performance function and the optimum algorithm are poorly defined.

We avoid the specification of any performance weighting function which is at best a subjective process and cannot be handled in an abstract study. However the parameters of the performance function can be treated on an objective basis. This requires that machines be specified in terms of gross characteristics. If the characterization of the machine is too fine then it is impossible to obtain a tractable abstraction and if the characterization is too gross then the abstraction is trivial.

It seems reasonable to first determine the effects of concurrency and parallelism on the rate of computation. If a single sequence machine executes a given algorithm in T time units, can m machines execute the algorithm in less than T/m time units? Further qualifications are necessary before a meaningful answer can be obtained for this question. Consider the usual flow diagram language used to describe algorithms for digital computation. The flow diagram consists of substitution statements which represent the direct computation and alternation statements which effect program control. A step in the computation is defined to be the time period required to execute a statement. We can now ask whether a multi-machine configuration can reduce the total number of statements of steps required to execute a given algorithm. A program for a multi-machine processor consists of a number of ordered program segments. The segments are ordered in the sense that a given segment requires input data computed by other segments. However, unless there exists a storage or time limitation, each of these segments can be executed in some sequence by a single machine. Thus a multi-machine configuration does not reduce the number of algorithm steps required to specify a given algorithm. A particular example of interest is look ahead. A second machine can be used at alteration points in the program to achieve look ahead. This results in an increase in the number of algorithm steps executed. A single sequence machine, under the assumption of equal probability at n alternation points, will on the average require n/2 additional look ahead computations. The multi-machine configuration will compute n look ahead steps, one for each alteration point.

A reduction in the time required for the computation is obtained by executing time independent sequences concurrently using a multi-machine configuration and by reducing the time period required to execute a given statement by the employment of concurrent or overlapping micro-operations within a given computer. If T_1 and T_2 are the average times required for non-look ahead and look ahead computational steps respectively, and if there are N non-look ahead steps, then the multi-machine computation period T_m in terms of the single sequence period T is

$$T_m = \frac{1 + \frac{n\tau_2}{N\tau_1}}{1 + \frac{n\tau_2}{2N\tau_2}} \frac{T}{m} > \frac{T}{m}$$

The optimum multi-machine configuration minimizes the time required for the computation and the number of machines required to obtain the computation in the minimal time period. It is clear that non-optimal assignments exist which employ more than the required minimal number of machines such that the number of machines can be reduced without changing the time required for the computation. In general, it is not possible to achieve 100 percent activity for all machines during the computation period. The actual number of machines which can be used is a function of the state of the computation.

The control of the multi-machine configuration adds steps to computation algorithm. Thus, when no space or time limitations exist it is not possible, even under the most ideal circumstances, to reduce the computation period of a multi-computer of m single-sequence machines to less than 1/m of the period required by a single sequence machine.

The previous discussion suggests the following problems: Studies of the quantity of control information required for multi-computer operation. Studies of finite automata with either space or time limitation or both.

4. Combinatorics and Switching Theory

4.0 Introduction

Most of the effort under this contract is being expended on "macroscopic" systems questions, e.g. problems of organization and control, and of the utilization of storage. Effort is also being given to questions of switching circuit synthesis which are "microscopic" in character. The justification for carrying on this latter type of research is two-fold. First, it is not always possible to totally separate the macroscopic issues from the microscopic ones. An intelligent evaluation of the practicality of a new computer organization usually requires an integrated outlook, which takes into account the fine structure of the system as well as its larger outlines.

Secondly, switching circuit theory is eclectic, drawing upon such mathematical topics as graph theory, matroid theory, semi-group and group theory, linear and dynamic programming, and classical combinatorial analysis. Techniques which are useful for solving switching circuit synthesis problems are usually sufficiently general to be useful for solving a variety of other combinatorial problems, including many having to do with macroscopic systems questions. Examples include combinatorial problems arising in the scheduling of multi-processor systems, the static and dynamic allocation of storage, and the assignment of capacities to data channels for maximum performance/cost ratio. It should thus be clear that our work in this area emphasizes basic "combinatorics" at least as much as "switching theory."

Our program in this area falls under the following project headings: (1) Covering Problems, (2) Threshold Networks, (3) Cellular Logic, (4) Sequential Circuit Synthesis, (5) Miscellaneous Problems. Project (1) is largely completed, and it is expected that project (2) will be completed in early 1966. Projects (3) and (4) are still in the formative stage. A description of each of these projects follows.

4.1 Covering Problems

It is generally acknowledged that the central combinatorial problem of switching theory is the so-called covering problem [all], which takes the form:

Minimize

$$cx = \sum_{j=1}^{n} c_j x_j \qquad (1)$$

subject to

$$Ax \geq 1 \qquad (2)$$

and

$$x_j = 0 \text{ or } 1 \quad (j=1,2,\ldots,n), \qquad (3)$$

where A is a (0,1)-matrix. (Each element a_{ij} of A is either 0 or 1). This problem arises in exactly this form in the second phase of the Quine method of Boolean minimization, and variants of it occur in a wide variety of other combinatorial contexts.

Research on covering problems has included:

(A) The identification of equivalent problems and special cases, and of relations between them.

(B) The investigation of the "graphic" covering problem, in which each column of A contains no more than two nonzero entries.

(C) The discovery of various duality relations, which hold for covering problems generally, and of a new method of solution suggested by these relations.

Results under heading (A) will be contained in a forthcoming report tentatively entitled, "Coverings, Packings, and Euler Lines." These include the following:

(i) consider constraints of the form

$$Ax \geq b \tag{2'}$$

and

$$x_j = \text{nonnegative integer} \tag{3'}$$

where A and b contain arbitrary nonnegative elements. Any problem of this apparently more general form is actually equivalent to a covering problem with constraints of the form (2) and (3).

(ii) For every "covering" problem of the form:

Minimize cx

subject to $Ax \geq b,$

$x_j = 0 \text{ or } 1,$

there is a complementary "packing" problem;

Maximize $c\bar{x}$

$A\bar{x} \leq \bar{b}$

$\overline{x_j} = 0 \text{ or } 1.$

The feasible and optimal solutions of the two problems are in one-one correspondence under the relation $x = 1-\bar{x}$.

(iii) We say that a covering problem or packing problem is "graphic" if its matrix A contains no more than one nonzero element in each column, and each nonzero element is 1. There is an obvious connection between these problems and problems which require that the vertices of a linear graph be covered with its edges (that at least b_i of the chosen edges be incident to vertex i) or that the graph be packed with its edges (no more than b_i of the chosen edges be incident to vertex i). Graphical covering problems and packing problems are, of course, complementary, in the sense of (ii). They are also equivalent to problems in which it is required to find a minimum-length tour which will pass through each edge of the graph at least once. This latter problem is solved by choosing a set of edges, such that when duplicated, an Euler line exists. Edmonds [b11] calls this problem the Chinese Postman's Problem.

A special case of the Chinese Postman's Problem is the following. What is a minimum-length input sequence, such that a given sequential machine will be forced through each possible state transition at least once? This is certainly a question of basic importance in the diagnosis of malfunctions in sequential circuits.

Results under heading (B)—investigation of graphic covering problems—are mainly in the explication and simplification of certain unpublished solution methods due to Edmonds. Edmonds devised an algebraically bounded computation for the Chinese Postman's Problem (a computation whose length grows only algebraically with the number of vertices in the graph) based on a computational method for packing problems which he calls "matching" problems.) These methods are, in turn, closely related to methods for solving the "shortest route" problem. Problems based upon directed graphs can also be handled without difficulty.

Work under heading (C)—duality relations—has been carried out in conjunction with National Science Foundation Grant GP 2778, "Partitioning Methods for Combinatorial Optimization." A few fundamental identities will serve to illustrate the duality properties in question. Define Cov A (the set of covers of A) and Cl A (the closure of A) as follows:

$$\text{Cov } A = \{x \mid Ax \geq 1,\ x_j = 0 \text{ or } 1\}$$

$$\text{Cl } A = \{a' \mid a' \geq A_i, \text{ for some } i, \text{ and } a_j' = 0 \text{ or } 1\}.$$

Then, for all A,B, such that A = Cl, B = Cl B,

$$\text{Cov Cov } A = A$$

$$\text{Cov } (A \cup B) = \text{Cov } A \cap \text{Cov } B$$

$$\text{Cov } (A \cap B) = \text{Cov } A \cup \text{Cov } B.$$

These relations may be compared with involution and DeMorgan's laws for sets:

$$\bar{\bar{A}} = A$$

$$(\overline{A \cup B}) = \bar{A} \cap \bar{B}$$

$$(\overline{A \cap B}) = \bar{A} \cup \bar{B}.$$

Computational methods suggested by these relations have been programmed for The University of Michigan IBM 7090 computer, and are currently being tested. A previously issued technical report [all] contains all theoretical results to date. This report has been accepted for publication in the SIAM Journal.

4.2 Threshold Networks

Work on networks of threshold elements is being carried out as the Ph.D. thesis project of R. Gonzalez, and should be completed by early 1966. This work is significant because of the new insights it contributes to the theory of linear inequalities, to nonlinear programming, and to "adaptive" networks of threshold elements, of the general type often proposed for pattern recognition problems. Research has included:

(i) The investigation of a "dual" method of elimination for solving systems of linear inequalities.

(ii) The study of minimal synthesis of two-level threshold networks, by an approach analogous to that used in the Quine method for minimal AND-OR synthesis.

(iii) The synthesis of economical "universal" networks which are capable of realizing any one of the 2^{2^n} switching functions of n variables by varying weights and thresholds.

(iv) The synthesis of universal networks for restricted families of switching functions, e.g. functions for which there are many "don't cares."

Investigation of the dual elimination method has shown that it is actually equivalent to the method of "double description" due to Motzkin, Raiffa, Thompson, and Thrall [b29], but much more easily derived. The importance of the method in the present context is, of course, that it provides an efficient method for testing the consistency of systems of linear inequalities, and thereby resolving the issue of linear separability for a given switching function.

Dual elimination provides the backbone of the synthesis method mentioned under (ii), and reported by Gonzalez and Lawler [a7]. This synthesis method is a two-phase method, just like the Quine method for Boolean minimization. The two phases are:

(a) generate the complete set of "best threshold approximations" of the switching function.

(b) select a minimal subset of best threshold approximations, such that they, together with one additional threshold element, are sufficient to realize the function.

Best threshold approximations, of course, correspond to prime implicants, and the selection problem (b) corresponds to the ordinary covering problem, but is more difficult to solve. Algorithms for two-level synthesis have been programmed for the IBM 7090 computer, and are currently being tested.

The synthesis of universal networks relies upon the application of the following theorem obtained by Gonzalez. We say that a subset of vertices of the n-cube is totally linearly separable if, for every possible partitioning of the subset into two parts, there exists a hyperplane which effects the partition.

Theorem

A given subset of $N + 1$ vertices $(x_0, x_1, x_2, \ldots, x_N)$ is totally linearly separable if and only if the N vectors $x_1 - x_0,\ x_2 - x_0,\ \ldots,\ x_N - x_0$ are linearly independent (the origin x_0 being chosen arbitrarily from among the subset of vertices).

In the case of problem (iv), the synthesis method requires that a minimum-rank partition of the "care" vertices be effected, such that each equivalence class under the partition is totally linearly separable. The problem this reduces essentially to a type studied in matroid theory [b10], and it appears to be possible to effect a direct application of known results.

4.3 Cellular Logic

It has become a truism that the emphasis of switching circuit synthesis should be changed to conform with the demands of modern integrated circuit technology; e.g. component counts should be de-emphasized and interconnections should be given primary attention. However, it appears that there are, as yet, no well-codified design requirements of the new technology. On the contrary, the situation is still fluid enough that many manufacturers would probably be willing to design their circuit layouts and interconnection wiring to accomodate a reasonable method of logical design.

An interesting and imaginative approach to this problem area is being taken by Minnick and others at Stanford Research Institute under the name "cellular logic" [b27]. Under this system, switching functions are realized by two-dimensional arrays of cells, where each cell can realize any one of several different functions of two variables, simply by cutting the appropriate "cutpoints" in the cell. All connections to these arrays are made at regular intervals along the edges.

A related approach is due to Canaday [b5], who proposes a two-dimensional array of 3-input "majority" elements. The cells of Canaday's arrays are simpler; however, he requires an entirely different—and probably more difficult—type of interconnection wiring.

Some of the questions which these proposals suggest are: What growth rates are necessary for the dimensions of these arrays as the number of switching variables increases? Can exponential growth be defeated in any way? What trade-offs are possible between the dimensions of the arrays and interconnection complexity? Are there any advantages to be gained by 3,4,...,N-dimensional arrays? (Consider the N-cube arrangement previously proposed for the Michigan Iterative Circuit Computer.) How should sequential circuits be realized by these arrays? (Minnick's cutpoint logic allows each cell to be a flip-flop, but systematic design methods have not yet been proposed.) What techniques can be used to circumvent the delays induced by many levels of logic?

4.4 Sequential Circuit Synthesis

Recent progress in the decomposition of finite automata includes the reformulation and simplification of the Krohn-Rhodes theory by Zeiger [b37], and new results on the reduction of feedback loops by Friedman [b13] and Brzozowski [b3]. A few tentative efforts have been made to apply these and other theoretical results to the synthesis of sequential circuits from a restricted set of simple modules. It seems not unreasonable that these efforts may eventually lead to efficient, systematic design methods for sequential circuits.

As in the case of cellular logic, plans for this project area are indefinite.

4.5 Miscellaneous Problems

Some minor effort has been devoted to combinatorial problems other than those described above. These include:

(1) Optimal encoding for the discrete noiseless channel with an alphabet whose symbols have unequal durations. This work was originally reported on at the International Conference on Microwaves, Circuit Theory, and Information Theory, Tokyo, Japan, October, 1964 [a9], and has now been revised and new tables for determining bounds on optimum encodings have been calculated.

(2) Optimal deferral scheduling for multiple channels and linear cost functions. This is an extension of previous results published in Management Science [a10]. It is shown that a dynamic programming method of solution exists and that such an optimal schedule for n jobs and m processors can be determined by an amount of computation which grows as n^m. This result depends on the linearity of the cost function.

Publications by Laboratory Personnel

[a1] Arnold, Richard F. and Richards, Donald L., "Monotone Reduction Algorithms," International Conference on Microwaves, Circuit Theory, and Information Theory, September 1964.

[a2] Arnold, Richard F. and Richards, Donald L., "Monotone Congruence Algorithms," Technical Report ISL-65-2, Information Systems Laboratory, Department of Electrical Engineering, The University of Michigan, Ann Arbor, April 1965.

[a3] Arnold, Richard F., "Random Access Storage Organization and Finite Automata," presented at the Rome Air Development Center-Hughes Aircraft Symposium on Logic, Computability and Automata, Rome, New York, August 1965, and to be published in the Proceedings, by Spartan Press.

[a4] Dauber, Philip S., "An Analysis of Errors in Finite Automata," Information and Control, 8 (1965), 295-303.

[a5] Dauber, Philip S., "An Analysis of Errors in Finite Automata," Technical Report ISL-65-1, Information Systems Laboratory, Department of Electrical Engineering, The University of Michigan, Ann Arbor, April 1965.

[a6] Dauber, Philip S., "Errors in Finite Automata," Ph.D. thesis, and Technical Report SEL-65-1, Systems Engineering Laboratory, Department of Electrical Engineering, The University of Michigan, Ann Arbor, October 1965.

[a7] Gonzalez, Rodolfo and Lawler, Eugene L., "Two-level Threshold Minimization," 1965 IEEE Conference on Switching Circuit Theory and Logical Design, Ann Arbor, October 1965, p. 94.

[a8] Lawler, Eugene L., "An Analysis of Roth's Methods of Synthesis," 7th Midwest Symposium on Circuit Theory, Ann Arbor (May 1964).

[a9] Lawler, Eugene L., "Combinatorial Aspects of Variable-Length Encoding," International Conference on Microwaves, Circuit Theory, and Information Theory, Tokyo, Japan, September 1964.

[a10] Lawler, Eugene L., "On Scheduling Problems with Deferral Costs," Management Science, 11, 2 (November 1964), 280-288.

[a11] Lawler, Eugene L., "Covering Problems: Duality Relations and a New Method of Solution," Technical Report ISL-65-3, Information Systems Laboratory, Department of Electrical Engineering, The University of Michigan, Ann Arbor, May 1965.

[al2] Reiter, Raymond, "A Study of a Model for Parallel Computation," Technical Report ISL-65-4, Information Systems Laboratory, Department of Electrical Engineering, The University of Michigan, Ann Arbor, July 1965.

[al3] Reiter, Raymond, "A Study of a Model for Parallel Computation II, Timing," to appear as a technical report of the Systems Engineering Laboratory, Department of Electrical Engineering, The University of Michigan, Ann Arbor.

[al4] Wood, David E. and Lawler, Eugene L., "Branch and Bound Algorithms," Technical Report, The University of Michigan 1965 Engineering Summer Conference on "Recent Mathematical Advances in Operations Research and the Management Sciences,".

Other References

[b1] Batson, Alan, "The Organization of Symbol Tables," Communications of the ACM (February 1965).

[b2] Bertier, P. and Roy, B., "Procedure de Resolution pour une Classe de Problems Pouvant Avoir un Caractere Combinatoire," Cahiers du Centre D'Etudes de Recherche Operationnelle, 6, (1964) 202-208.

[b3] Brzozowski, J. A., "On Single-Loop Realizations of Automata," IEEE Conference Record on Switching Circuit Theory and Logical Design, New York (October 1965).

[b4] Burks, A. W., Warren, P. W., and Wright, J. B., "An Analysis of a Logical Machine Using Parenthesis-Free Notation," Math. Tables and Other Aids to Computation, 7 (1954), 53-57.

[b5] Canaday, R. H., "Two-Dimensional Iterative Logic," Report ESL-R-210, Electronic Systems Laboratory, Massachusetts Institute of Technology, September 1964.

[b6] Chomsky, N., "Formal Properties of Grammars," in Handbook of Mathematical Psychology, John Wiley and Sons, Inc., New York, 2 (1963), 323-418.

[b7] Chomsky, N., "On Certain Formal Properties of Grammars," Information and Control, 2 (1959), 137-167.

[b8] Dantzig, G. B., Fulkerson, D. R., and Johnson, S., "Solution of a Large-Scale Travelling Salesman Problem," Journal of Operations Research Society of America, 2 (1954), 393-410.

[b9] Davis, M. D., Computability and Unsolvability, McGraw-Hill Book Company, New York (1956).

[b10] Edmonds, J., "Minimum Partition of a Matroid into Independent Subsets," J. Research of the NBS, Section B., Mathematics and Mathematical Physics 69B, 1,2 (January-June 1965), 67.

[b11] Edmonds, J., "Paths, Trees and Flowers, Canadian J. Math. (May 1965).

[b12] Evey, R. J., "Applications of Pushdown-Store Machines," AFIPS Conference Proceedings, 24 (1963), 215-228.

[b13] Friedman, A. D., "Feedback in Asynchronous Circuits," 1965 IEEE Conference Record on Switching Circuit Theory and Logical Design, New York (October 1965), 94.

[b14] Gilbert, E., and Moore, E., "Variable-Length Binary Encoding," Bell Systems Technical Journal, 38, (1959), 933-967.

[b15] Ginsburg, S., A Survey of ALGOL-Like and Context-Free Language Theory, System Development Corporation, Report No. TM-738/006/00, March 6, 1964.

[b16] Ginsburg, S., and Rice, H. G., "Two Families of Languages Related to ALGOL," Journal of the ACM, 9 (1962), 350-371.

[b17] Ginsburg, S., and Rose, G. F., "Some Recursively Unsolvable Problems in ALGOL-Like Languages," Journal of the ACM, 10 (1963), 29-47.

[b18] Harrison, M. A., "On the Error Correcting Capacity of Finite Automata," Information and Control, 8 (1965), 430-450.

[b19] Hartmanis, J., "Loop-free Structure of Sequential Machines," Information and Control, 5, (March 1962), 25-43.

[b20] Hartmanis, J. and Stearns, R. E., "A Study of Feedback and Errors in Sequential Machines," IEEE Trans. on Electr. Computers, EC-12, (1963) 223-232.

[b21] Hartmanis, J., and Stearns, R. E., "Computational Complexity of Recursive Algorithms," in Proceedings of the Fifth Annual Symposium on Switching Theory and Logical Design, Princeton, New Jersey (1964), 82-90.

[b22] Hartmanis, J. and Stearns, R. E., "Pair Algebra and its Application to Automata Theory," Information and Control, 7 (December, 1964).

[b23] Jensen, P. A., "Bibliography on Redundancy Techniques," Redundancy Techniques in Computing Systems (edited by R. H. Wilcox and W. C. Mann), Spartan Books (1962), 389-403.

[b24] Karp, R. M., and Miller, R. E., "Properties of a Model for Parallel Computations: Determinacy, Termination, Queueing," IBM Research Paper RC-1285. (To appear in SIAM Journal).

[b25] Krohn, K. B. and Rhodes, J. L., "Algebraic Theory of Machines," Mathematical Theory of Automata, Polytechnic Press, Brooklyn, New York (1963).

[b26] Medvedev, Y. T., "On the Class of Events Representable in a Finite Automaton," Sequential Machines: Selected Papers (E. F. Moore, ed.), Addison-Wesley, Reading Massachusetts (1964).

[b27] Minnick, R. C., "Cutpoint Cellular Logic," IEEE Trans. on Electr. Computers, EC-13, 6 (December 1964), 685-698.

[b28] Moore, E. F., and Shannon, C. E., "Reliable Circuits Using Less Reliable Relays," Journal of the Franklin Institute, 262 (1956) 191-208, 281-297.

[b29] Motzkin, T., Raiffa, S. H., Thompson, G. L, and Thrall, R. M., "The Double Description Method," Contributions to the Theory of Games, Vol. II (eds., H. W. Kuhn, A. W. Tucker) Princeton University Press, Princeton, New Jersey (1953).

[b30] Naur, P., et al., "Revised Report on the Algorithmic Language ALGOL 60," Journal of the ACM, 9, (1962), 350-371.

[b31] Neumann, P. G., "Error Limiting Coding Using Information Loss-less Machines," IEEE Trans. on Information Theory, IT-10, (1964), 108-115.

[b32] Perles, M., Rabin, M. O., and Shamir, E., "The Theory of Definite Automata," IEEE Trans. on Electr. Computers, EC-12, (June 1963), 233-242.

[b33] Pontryagin, L. S., The Mathematical Theory of Optimal Processes, Interscience Publishers, New York (1962).

[b34] Schutzenberger, M. P., "On Context-Free Languages and Pushdown Automata," Information and Control, 6 (1963), 246-264.

[b35] von Neumann, J., "Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components," Automata Studies (Ed. by C. E. Shannon and J. McCarthy), Princeton University Press, Princeton, New Jersey (1956), 43-98.

[b36] Winograd, S., "Input Error Limiting Automata," Journal of the ACM, 11, (1964), 338-361.

[b37] Zeiger, P., "Loop-free Decomposition of Sequential Machines," Ph.D. Thesis, Massachusetts Institute of Technology (1964).

UNCLASSIFIED
Security Classification

DOCUMENT CONTROL DATA - R&D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)* Systems Engineering Laboratory Department of Electrical Engineering The University of Michigan Ann Arbor, Michigan	2a. REPORT SECURITY CLASSIFICATION Unclassified 2b. GROUP

3. REPORT TITLE

Mathematical Models of Information Systems

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*

Interim

5. AUTHOR(S) *(Last name, first name, initial)*

Richard Arnold	Richard Karp
Harvey Garner	Eugene L. Lawler

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
April 1966	84	51

8a. CONTRACT OR GRANT NO. AF30(602)-3546	9a. ORIGINATOR'S REPORT NUMBER(S)
b. PROJECT NO. 5581	
c. Task No. 558109	9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*
d.	RADC-TR-66-37

10. AVAILABILITY/LIMITATION NOTICES

This document is subject to special export controls and each transmittal to foreign governments or foreign nationals may be made only with prior approval of RADC (EMLI), GAFB, N.Y. 13440.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY Rome Air Development Center (EMIID) Griffiss Air Force Base, N.Y. 13440.

13. ABSTRACT

This report is the first interim report of a three year study and investigation by the University of Michigan. The primary objective of this effort is the study and development of mathematical models of information processing systems. The general area of research includes machine design, automata theory, and the application of mathematical models to problems in machine design. The areas of research in this report are divided into these four areas (1) Automata Theory and Applications, (2) Theory of Algorithms, (3) System Analysis, and (4) Combinatorics and Switching Theory.

DD FORM 1 JAN 64 1473

UNCLASSIFIED
Security Classification

UNCLASSIFIED
Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Data Processing Systems Mathematical Analysis Computer Logic						

INSTRUCTIONS

1. ORIGINATING ACTIVITY: Enter the name and address of the contractor, subcontractor, grantee, Department of Defense activity or other organization (*corporate author*) issuing the report.

2a. REPORT SECURITY CLASSIFICATION: Enter the overall security classification of the report. Indicate whether "Restricted Data" is included. Marking is to be in accordance with appropriate security regulations.

2b. GROUP: Automatic downgrading is specified in DoD Directive 5200.10 and Armed Forces Industrial Manual. Enter the group number. Also, when applicable, show that optional markings have been used for Group 3 and Group 4 as authorized.

3. REPORT TITLE: Enter the complete report title in all capital letters. Titles in all cases should be unclassified. If a meaningful title cannot be selected without classification, show title classification in all capitals in parenthesis immediately following the title.

4. DESCRIPTIVE NOTES: If appropriate, enter the type of report, e.g., interim, progress, summary, annual, or final. Give the inclusive dates when a specific reporting period is covered.

5. AUTHOR(S): Enter the name(s) of author(s) as shown on or in the report. Enter last name, first name, middle initial. If military, show rank and branch of service. The name of the principal author is an absolute minimum requirement.

6. REPORT DATE: Enter the date of the report as day, month, year; or month, year. If more than one date appears on the report, use date of publication.

7a. TOTAL NUMBER OF PAGES: The total page count should follow normal pagination procedures, i.e., enter the number of pages containing information.

7b. NUMBER OF REFERENCES: Enter the total number of references cited in the report.

8a. CONTRACT OR GRANT NUMBER: If appropriate, enter the applicable number of the contract or grant under which the report was written.

8b, 8c, & 8d. PROJECT NUMBER: Enter the appropriate military department identification, such as project number, subproject number, system numbers, task number, etc.

9a. ORIGINATOR'S REPORT NUMBER(S): Enter the official report number by which the document will be identified and controlled by the originating activity. This number must be unique to this report.

9b. OTHER REPORT NUMBER(S): If the report has been assigned any other report numbers (*either by the originator or by the sponsor*), also enter this number(s).

10. AVAILABILITY/LIMITATION NOTICES: Enter any limitations on further dissemination of the report, other than those imposed by security classification, using standard statements such as:

(1) "Qualified requesters may obtain copies of this report from DDC."

(2) "Foreign announcement and dissemination of this report by DDC is not authorized."

(3) "U. S. Government agencies may obtain copies of this report directly from DDC. Other qualified DDC users shall request through
_______________________."

(4) "U. S. military agencies may obtain copies of this report directly from DDC. Other qualified users shall request through
_______________________."

(5) "All distribution of this report is controlled. Qualified DDC users shall request through
_______________________."

If the report has been furnished to the Office of Technical Services, Department of Commerce, for sale to the public, indicate this fact and enter the price, if known.

11. SUPPLEMENTARY NOTES: Use for additional explanatory notes.

12. SPONSORING MILITARY ACTIVITY: Enter the name of the departmental project office or laboratory sponsoring (*paying for*) the research and development. Include address.

13. ABSTRACT: Enter an abstract giving a brief and factual summary of the document indicative of the report, even though it may also appear elsewhere in the body of the technical report. If additional space is required, a continuation sheet shall be attached.

It is highly desirable that the abstract of classified reports be unclassified. Each paragraph of the abstract shall end with an indication of the military security classification of the information in the paragraph, represented as (*TS*), (*S*), (*C*), or (*U*).

There is no limitation on the length of the abstract. However, the suggested length is from 150 to 225 words.

14. KEY WORDS: Key words are technically meaningful terms or short phrases that characterize a report and may be used as index entries for cataloging the report. Key words must be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location, may be used as key words but will be followed by an indication of technical context. The assignment of links, rules, and weights is optional.

UNCLASSIFIED
Security Classification

SUPPLEMENTARY

INFORMATION

NOTICES OF CHANGES IN CLASSIFICATION, DISTRIBUTION AND AVAILABILITY

69-20 15 October 1969

IDENTIFICATION	FORMER STATEMENT	NEW STATEMENT	AUTHORITY
AD-483 281 University of Michigan, Ann Arbor. Systems Engineering Lab. Interim rept. no. 1. Rept. no. RADC TR-66-37 Apr 66 Contract AF 30(602)-3546	No Foreign without approval of Rome Air Development Center, Attn: EMLI, Griffiss AFB, N. Y.	No limitation	RADC, USAF ltr, 14 Jul 69